Learning Transferable Architectures for Scalable Image Recognition

Zoph et al., CVPR ‘18

Presented By: Kyle Leeners, Mir Rayat Imtiaz Hossain, Nam Hee Kim, Chris Yoon

Table of Contents

Introduction

- Scalable method to optimize CNNs
    - Relatively low compute
    - Easily generalizable
- Neural Architecture search (NAS)
    - Reinforcement learning to optimize network architectures
    - Expensive on large dataset (ImageNet)
- Introduce constraints
    - Train on proxy dataset (CIFAR-10)
        - Smaller = Reduced compute time
    - Architecture complexity independent of network depth and image size
        - Architecture cells have same structure but different weights
- Constraints accelerate search speed on CIFAR-10 by a factor of 7x

Related Work

- Hyperparameter optimization
    - Neural Fabrics : A “fabric” that embeds an exponentially large number of architectures
    - DiffRNN: Gradient descent on the number of neurons
    - MetaQNN: Build CNN via reinforcement learning
    - DeepArchitecture: Develop tree-structured search spaces over network architectures and hyperparameters
    - Evolutionary algorithms: Not much success at large scale
- Search Space
    - Much inspiration from LSTMs
    - NAS: Use RNN trained via RL to generate neural networks
- Transfer learning
    - Xie and Yuille: Transfer learning between CIFAR-10 and ImageNet but performance is normally below state-of-the-art

Related Work

- Meta-learning
  - Much attention in recent years but most approaches have not been scaled to large problems like ImageNet
  - Recent work by Wichrowska et. al. has had some success in learning an optimizer for ImageNet classification that achieved notable improvements
- Modular Structure of Convolutional Cell
  - VGG
  - Inception
  - ResNet
  - Xception/MobileNet

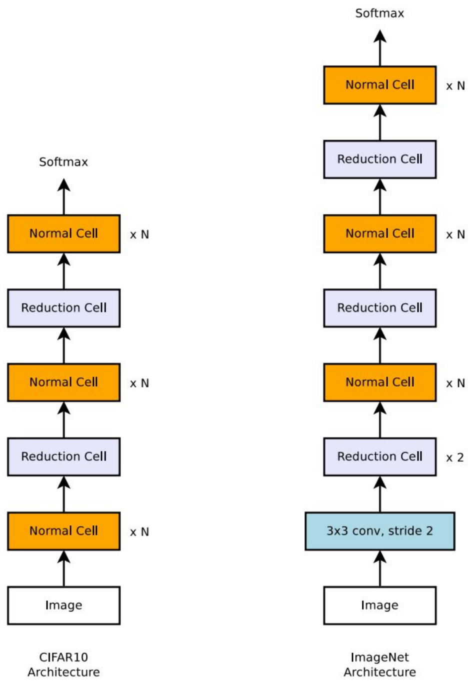
Proposed Method

- Use a search method to find appropriate CNN architecture on a dataset.
- The main contribution of the paper is to define the search space.
- Motivation of search space definition:
  - Most state-of-the-art networks repeat a certain pattern of architecture.

The search space

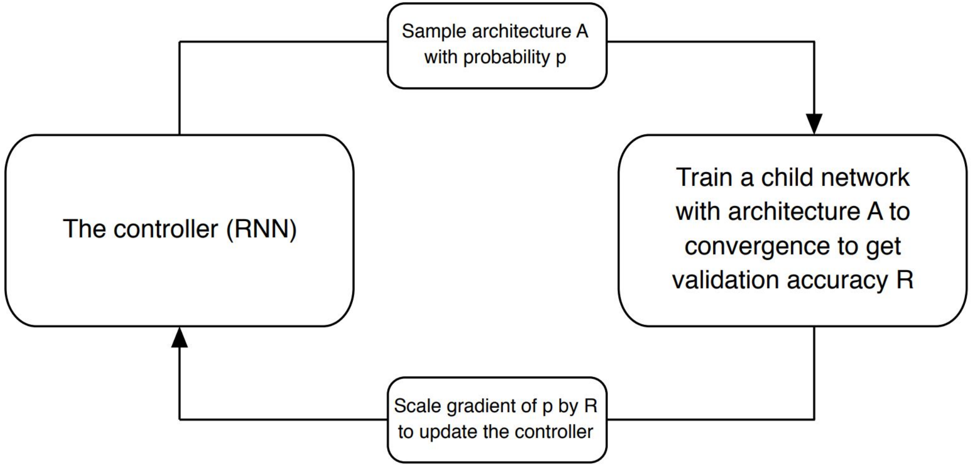
- Define predetermined set of operations / architectures.
- Compose and combine them to form a “convolutional cell”.
- Stack the “convolutional cells”, each having different weights.
- Two types of convolutional cells:
  - Normal Cell: Same feature dimension as input
  - Reduction Cell: Reduce the feature size by 2.

- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv

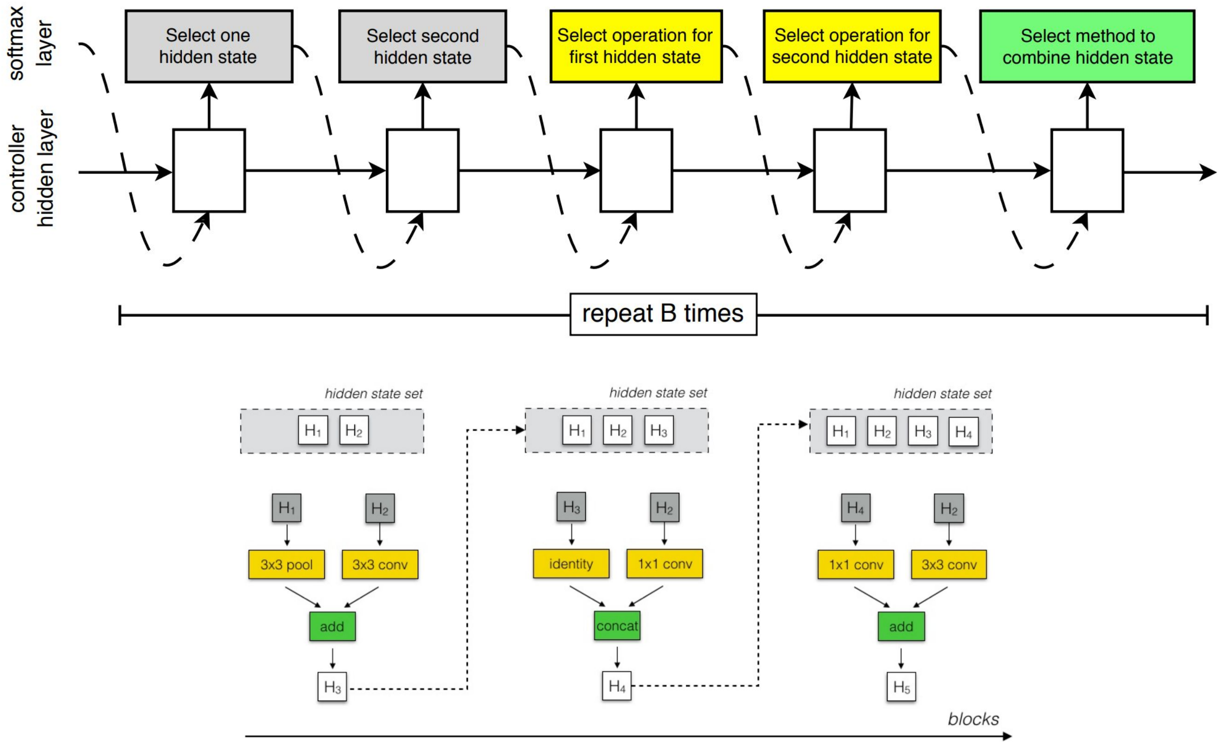


Search Method

- Used Network Architecture Search (NAS) (Zoph and Le, ICLR 2017).



Search Method



Experiments

1 In this section, we describe our experiments with the method described above to learn convolutional cells. In summary, all architecture searches are performed using the CIFAR-10 classification task [31]. The controller RNN was trained using Proximal Policy Optimization (PPO) [51] by employing a global workqueue system for generating a pool of child networks controlled by the RNN. In our experiments, the pool of workers in the workqueue consisted of 500 GPUs.

2 The result of this search process over 4 days yields several candidate convolutional cells. We note that this search procedure is almost $7\times$ faster than previous approaches [71] that took 28 days.[1] Additionally, we demonstrate below that the resulting architecture is superior in accuracy.

3 pendix B) and report their results as well. We call the three networks constructed from the best three searches *NASNet-A*, *NASNet-B* and *NASNet-C*.

We demonstrate the utility of the convolutional cells by employing this learned architecture on CIFAR-10 and a family of ImageNet classification tasks. The latter family of tasks is explored across a few orders of magnitude in computational budget. 4 After having learned the convolutional cells, several hyper-parameters may be explored to build a final network for a given task: (1) the number of cell repeats N and (2) the number of filters in the initial convolutional cell. 5 After selecting the number of initial filters, we use a common heuristic to double the number of filters whenever the stride is 2. Finally, we define a simple notation, e.g., 4 @ 64, to indicate these two parameters in all networks, where 4 and 64 indicate the number of cell repeats and the number of filters in the penultimate layer of the network, respectively.

"Best Architecture": NASNet-A

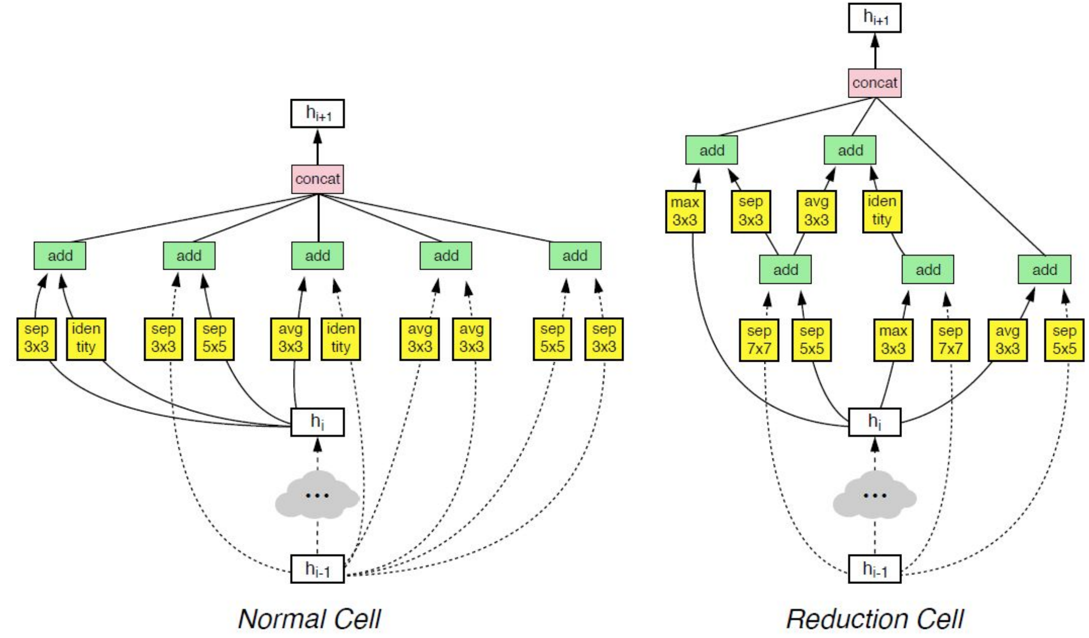


Figure 4. Architecture of the best convolutional cells (NASNet-A) with $B = 5$ blocks identified with CIFAR-10 . The input (white) is the hidden state from previous activations (or input image). The output (pink) is the result of a concatenation operation across all resulting branches. Each convolutional cell is the result of B blocks. A single block is corresponds to two primitive operations (yellow) and a combination operation (green). Note that colors correspond to operations in Figure 3.

CIFAR-10 **Classification** Benchmark

model	depth	# params	error rate (%)
DenseNet ($L = 40, k = 12$) [26]	40	1.0M	5.24
DenseNet($L = 100, k = 12$) [26]	100	7.0M	4.10
DenseNet ($L = 100, k = 24$) [26]	100	27.2M	3.74
DenseNet-BC ($L = 100, k = 40$) [26]	190	25.6M	3.46
Shake-Shake 26 2x32d [18]	26	2.9M	3.55
Shake-Shake 26 2x96d [18]	26	26.2M	2.86
Shake-Shake 26 2x96d + cutout [12]	26	26.2M	2.56
NAS v3 [71]	39	7.1M	4.47
NAS v3 [71]	39	37.4M	3.65
NASNet-A (6 @ 768)	-	3.3M	3.41
NASNet-A (6 @ 768) + cutout	-	3.3M	2.65
NASNet-A (7 @ 2304)	-	27.6M	2.97
NASNet-A (7 @ 2304) + cutout	-	27.6M	2.40
NASNet-B (4 @ 1152)	-	2.6M	3.73
NASNet-C (4 @ 640)	-	3.1M	3.59

previous best

new best

Table 1. Performance of Neural Architecture Search and other state-of-the-art models on CIFAR-10. All results for NASNet are the mean accuracy across 5 runs.

ImageNet **Classification** Benchmark

NASNet-A beats all other models with **fewer parameters!**

Learned on CIFAR-10

Model	image size	# parameters	Mult-Adds	Top 1 Acc. (%)	Top 5 Acc. (%)
Inception V2 [29]	224×224	11.2 M	1.94 B	74.8	92.2
NASNet-A (5 @ 1538)	**299×299**	**10.9 M**	**2.35 B**	**78.6**	**94.2**
Inception V3 [60]	299×299	23.8 M	5.72 B	78.8	94.4
Xception [9]	299×299	22.8 M	8.38 B	79.0	94.5
Inception ResNet V2 [58]	299×299	55.8 M	13.2 B	80.1	95.1
NASNet-A (7 @ 1920)	**299×299**	**22.6 M**	**4.93 B**	**80.8**	**95.3**
ResNeXt-101 (64 x 4d) [68]	320×320	83.6 M	31.5 B	80.9	95.6
PolyNet [69]	331×331	92 M	34.7 B	81.3	95.8
DPN-131 [8]	320×320	79.5 M	32.0 B	81.5	95.8
SENet [25]	**320×320**	**145.8 M**	**42.3 B**	**82.7**	**96.2**
NASNet-A (6 @ 4032)	**331×331**	**88.9 M**	**23.8 B**	**82.7**	**96.2**

Table 2. Performance of architecture search and other published state-of-the-art models on ImageNet classification. Mult-Adds indicate the number of composite multiply-accumulate operations for a single image. Note that the composite multiple-accumulate operations are calculated for the image size reported in the table. Model size for [25] calculated from open-source implementation.

ImageNet **Classification** Benchmark

NASNet-A beats all other models with **fewer-ish operations!**

Model	# parameters	Mult-Adds	Top 1 Acc. (%)	Top 5 Acc. (%)
Inception V1 [59]	6.6M	1,448 M	69.8 †	89.9
MobileNet-224 [24]	4.2 M	569 M	70.6	89.5
ShuffleNet (2x) [70]	$\sim$ 5M	524 M	70.9	89.8
NASNet-A (4 @ 1056)	**5.3 M**	**564 M**	**74.0**	**91.6**
NASNet-B (4 @ 1536)	5.3M	488 M	72.8	91.3
NASNet-C (3 @ 960)	4.9M	558 M	72.5	91.0

Table 3. Performance on ImageNet classification on a subset of models operating in a constrained computational setting, i.e., < 1.5 B multiply-accumulate operations per image. All models use 224x224 images. † indicates top-1 accuracy not reported in [59] but from open-source implementation.

COCO **Object Detection** Benchmark

NASNet-A beats all other models

Model	**resolution**	**mAP (mini-val)**	**mAP (test-dev)**
MobileNet-224 [24]	600 × 600	19.8%	-
ShuffleNet (2x) [70]	600 × 600	24.5%†	-
NASNet-A (4 @ 1056)	600 × 600	**29.6%**	-
ResNet-101-FPN [36]	800 (short side)	-	36.2%
Inception-ResNet-v2 (G-RMI) [28]	600 × 600	35.7%	35.6%
Inception-ResNet-v2 (TDM) [52]	600 × 1000	37.3%	36.8%
NASNet-A (6 @ 4032)	800 × 800	41.3%	40.7%
NASNet-A (6 @ 4032)	1200 × 1200	**43.2%**	**43.1%**
ResNet-101-FPN (RetinaNet) [37]	800 (short side)	-	39.1%

Table 4. Object detection performance on COCO on *mini-val* and *test-dev* datasets across a variety of image featurizations. All results are with the Faster-RCNN object detection framework [47] from a single crop of an image. Top rows highlight mobile-optimized image featurizations, while bottom rows indicate computationally heavy image featurizations geared towards achieving best results. All *mini-val* results employ the same 8K subset of validation images in [28].

Discussion: Pros & Cons

Pros

- High impact paper - less of grad student/researcher descent
- Thorough evaluations and comparisons to appropriate baselines
- Architecture transferability to other tasks
- Inspired many impactful future works

Cons/Questions

- Block composition is manually determined
- Still expensive
- Faster than NAS...but at what cost?
  - Assume a block can be learned on smaller dataset
  - Restricted operations
- Gap between RL and Random is small (although section 4.4 address this)
- Why does ScheduledDropPath work better?

Discussion: Future work

Efficient Neural Architecture Search via Parameter Sharing (16 hrs on 1080ti)

- Key Idea: Don’t retrain weights during the search - share them!
- 2.89% test error vs 2.65% test error

N2N LEARNING: NETWORK TO NETWORK COMPRESSION VIA POLICY GRADIENT REINFORCEMENT LEARNING (? hrs 4 Titan X)

- RNN to select: Stage 1) layers to remove Stage 2) channels to remove

ResNet-34	Teacher	92.05%	21.28M	—	—
	Student (Stage1)	93.54%	3.87M	+1.49%	5.5x
	Student (Stage1+Stage2)	92.35%	2.07M	+0.30%	10.2x

Discussion: Future work

AMC: AutoML for Model Compression and Acceleration on Mobile Devices (fastest is 1 hr on Titan Xp)

	NAS	NT	N2N	**AMC**
optimize for accuracy	✓	✓	✓	✓
optimize for latency				✓
simple, non-RNN controller				✓
fast exploration with few GPUs		✓	✓	✓
continuous action space				✓

ResNet-50 (93.53%)	**AMC** ($R_{\mathbf{Param}}$)	60% Params	**93.64**	**93.55**	-

Thank you